COVER SHEET

Case No. **WHOVIS.018PR2**
Date: April 27, 2000

ASSISTANT COMMISSIONER FOR PATENTS
WASHINGTON, D.C. 20231

ATTENTION: PROVISIONAL PATENT APPLICATION

Sir:

This is a request for filing a PROVISIONAL APPLICATION FOR PATENT under 37 CFR § 1.53(c).

For: **SECURE SITE FOR INTERNET TRANSACTIONS**

Enclosed are:

(X) Specification in 21 pages.

(X) A return prepaid postcard.

Was this invention made by an agency of the United States Government or under a contract with an agency of the United States Government?

(X) No.

( ) Yes. The name of the U.S. Government agency and the Government contract number are:

(X) Please send correspondence to:

John R. King
Knobbe, Martens, Olson & Bear, LLP
620 Newport Center Dr., 16th Floor
Newport Beach, CA 92660

Respectfully submitted,

John M. Grover
Registration No. 42,610

H:\DOCS\JRK\JRK-3261.DOC

KNOBBE, MARTENS, OLSON & BEAR
A LIMITED LIABILITY PARTNERSHIP INCLUDING
PROFESSIONAL CORPORATIONS

PATENT, TRADEMARK AND COPYRIGHT CAUSES

620 NEWPORT CENTER DRIVE
SIXTEENTH FLOOR
NEWPORT BEACH, CALIFORNIA 92660-8016
(949) 760-0404
FAX (949) 760-9502
INTERNET WWW.KMOB.COM

KNOBBE*
MARTENS*
OLSON*
BEAR
L. OLSON*
BUNKER
NIEMAN
ROSE
J. LESNIAK
NED A. ISRAELSEN
DREW S. HAMILTON
JERRY T. SEWELL
JOHN B. SGANGA, JR
EDWARD A. SCHLATTER
GERARD VON HOFFMANN
JOSEPH R. RE
CATHERINE J. HOLLAND
JOHN M. CARSON
KAREN VOGEL WEIL
ANDREW H. SIMPSON
JEFFREY L. VAN HOOSEAR
DANIEL E. ALTMAN
MARGUERITE L. GUNN
STEPHEN C. JENSEN
VITO A. CANUSO III
WILLIAM H. SHREVE
LYNDA J. ZADRA-SYMES†
STEVEN J. NATAUPSKY
PAUL A. STEWART
JOSEPH F. JENNINGS
CRAIG S. SUMMERS
ANNEMARIE KAISER

BRENTON R. BABCOCK
THOMAS F. SMEGAL, JR
MICHAEL H. TRENHOLM
DIANE M. REED
JONATHAN A. BARNEY
RONALD J. SCHOENBAUM
JOHN R. KING
FREDERICK S. BERRETTA
NANCY WAYS VENSKO
JOHN P. GIEZENTANNER
ADEEL S. AKHTAR
GINGER R. DREGER
THOMAS R. ARNO
DAVID N. WEISS
DANIEL HART, PH.D.
DOUGLAS G. MUEHLHAUSER
LORI LEE YAMATO
MICHAEL K. FRIEDLAND
STEPHEN M. LOBBIN
STACEY R. HALPERN
DALE C. HUNT, PH.D.
LEE W. HENDERSON, PH.D.
DEBORAH S. SHEPHERD
RICHARD E. CAMPBELL
MARK M. ABUMERI
JON W. GURKA
ERIC M. NELSON
MARK R. BENEDICT, PH.D.
PAUL N. CONOVER
ROBERT J. ROBY
SABING H. LEE
KAROLINE A. DELANEY

JOHN W. HOLCOMB
JAMES J. MULLEN, III, PH.D.
JOSEPH S. CIANFRANI
JOSEPH M. REISMAN, PH.D.
WILLIAM R. ZIMMERMAN
GLEN L. NUTTALL
ERIC S. FURMAN, PH.D.
TIRZAH ABE LOWE
GEOFFREY Y. IIDA
ALEXANDER S. FRANCO
SANJIVPAL S. GILL
SUSAN M. MOSS
JAMES W. HILL, M.D.
ROSE M. THIESSEN, PH.D.
MICHAEL L. FULLER
MICHAEL A. GUILIANA
MARK J. KERTZ
RABINDER N. NARULA
BRUCE S. ITCHKAWITZ, PH.D.
PETER M. MIDGLEY
THOMAS S. MCCLENAHAN
MICHAEL S. OKAMOTO
JOHN M. GROVER
MALLARY K. DE MERLIER
IRFAN A. LATEEF
AMY C. CHRISTENSEN
SHARON S. NG
MARK J. GALLAGHER, PH.D.
DAVID G. JANKOWSKI, PH.D.
BRIAN C. HORNE
PAYSON J. LEMEILLEUR
WILLIAM G. BERRY

OF COUNSEL
JERRY R. SEILER

JAPANESE PATENT ATTY
KATSUHIRO ARAI**

EUROPEAN PATENT ATTY
MARTIN HELLEBRANDT

KOREAN PATENT ATTY
MINCHEOL KIM

SCIENTISTS & ENGINEERS
(NON-LAWYERS)

RAIMOND J. SALENIEKS**
NEIL S. BARTFELD, PH.D.**
DANIEL E. JOHNSON, PH.D.**
JEFFERY KOEPKE, PH.D.**
KHURRAM RAHMAN, PH.D.
JENNIFER A. HAYNES, PH.D.
BRENDAN P. O'NEILL, PH.D.
THOMAS Y. NAGATA
ALAN C. GORDON
LINDA H. LIU
YASHWANT VAISHNAV, PH.D.
MEGUMI TANAKA
ANDREW N. MERICKEL
CHE S. CHERESKIN, PH.D.**
JASON I. SAATHOFF
ERIK W. ARCHBOLD
PHILIP C. HARTSTEIN
JULIE A. HOPPER
CHRIS S. CASTLE

* A PROFESSIONAL CORPORATION
† ALSO BARRISTER AT LAW (U.K.)
** U.S. PATENT AGENT

Assistant Commissioner for Patents
Washington, D.C. 20231

CERTIFICATE OF MAILING BY "EXPRESS MAIL"

Attorney Docket No.	:	WHOVIS.018PR2
Applicants	:	Dickinson, et al.
For	:	SECURE SITE FOR INTERNET TRANSACTIONS
Attorney	:	John R. King
"Express Mail" Mailing Label No.	:	EL 512 369 644 US
Date of Deposit	:	April 27, 2000

I hereby certify that the accompanying

Transmittal in Duplicate; Specification in 21 pages; Return Prepaid Postcard

are being deposited with the United States Postal Service "Express Mail Post Office to Addressee" service under 37 CFR 1.10 on the date indicated above and are addressed to the Assistant Commissioner for Patents, Washington, D.C. 20231.

Donald L. King

H:\DOCS\RK\RK-3262.DOC:sb

201 CALIFORNIA STREET
SUITE 1150
SAN FRANCISCO, CALIFORNIA 94111
(415) 954-4114
FAX (415) 954-4111

501 WEST BROADWAY
SUITE 1400
SAN DIEGO, CALIFORNIA 92101
(619) 235-8550
FAX (619) 235-0176

3801 UNIVERSITY AVENUE
SUITE 710
RIVERSIDE, CALIFORNIA 92501
(909) 781-9231
FAX (909) 781-4507

1875 CENTURY PARK EAST
SUITE 600
LOS ANGELES, CALIFORNIA 90067
(310) 407-5484
FAX (310) 407-5485

Secure Site For Internet Transactions

The following materials are part of, and together form, this provisional patent application, titled, "Secure Site For Internet Transactions," filed on April 25, 2000.

List of Materials:

1. Core Uses of Public Key Technology
2. Server-Centric Private Key Processing
3. Who? Vision Authentication System
4. Signing Data Flow
5. SignEngine
6. Depository Process
7. Features of the TrustEngine Technology
8. Security Aspects of the ethenticate™ System

H:\DOCS\JMG-1399.DOC
04/24/2000

Core Uses of Public Key Technology

The purpose of this note is to describe the three fundamental applications of public key technology in a public key infrastructure (PKI).

Public key technology has many interesting applications. Most of them fall into one of three broad categories: key management, signature, and authentication. A successful PKI will take advantage of all three. We will describe how public key technology is used in these situations.

In public key technology, keys are generated in pairs. The two keys are bound together in the following way: what one key encrypts, only the other key can decrypt. This key pair is owned by an entity, typically an individual or computer. One of the two keys is designated as the public key and the other as the private key. As the name suggests, a public key can be made public without compromising security. The public key along with identifying information of the key pair owner is placed into what is called a certificate. This certificate may then be stored in a public database or directory. However, the private key is known only to the entity who *owns* the key pair. The underlying assumption here is that the private key cannot be derived from the public key. This is a fundamental difference from symmetric key technology where a decryption key is easily derived from an encryption key. As an aside, public key technology also goes by the name of asymmetric key technology. In the remainder of this note we will discuss how applications make use of public key technology.

Public key technology involves rather complicated mathematical computations. Compared to symmetric key technology (which uses algorithms such as DES or RC4), public key technology is quite slow. For that reason data encryption is typically done using symmetric key technology. However, these symmetric keys need to be delivered securely to the entities wishing to decrypt the data. Public key technology can be used to deliver these keys and this key delivery process is called key management.

Key management applications that use public key technology are abundant. The list includes secure e-mail, SSL (secure sockets layer used in browsers), and the IPSec standard (used to secure VPNs). Let's briefly describe how key management is handled using public key technology. We'll use e-mail as an example. Say Greg wants to send an encrypted e-mail to Michelle. Let's say he has access to her certificate which holds her public key. Greg encrypts the e-mail with a symmetric key and then encrypts the symmetric key with Michelle's public key. He sends Michelle the encrypted e-mail together with the encrypted symmetric key. When Michelle receives this, she has everything she needs to read the e-mail. The first thing she does is get the symmetric key. She does this by decrypting the encrypted symmetric key with her private key. Now that she has the symmetric key, she simply decrypts the e-mail message. That's key management in a nutshell.

The distinguishing feature of key management applications is that Michelle's public key was used by someone else to encrypt a symmetric key for her use. Michelle then used her private key to decrypt and gain access to the symmetric key. In signature and authentication applications this model is turned upside down.

Suppose Michelle has a document she wishes to sign. The goal here is to have proof that Michelle and only Michelle signed the document. This can be done using public key

technology. Since public key technology is relatively slow and documents can be arbitrarily large, Michelle first hashes the document. The resulting hash value is a block of 20 bytes of random looking data. We can think of this hash value as a fingerprint of the document, since in practice no two documents will ever have the same hash value. Michelle then uses her private key to encrypt this hash value. This encrypted hash value forms Michelle's signature of the document.

Let's describe how we would verify Michelle's signature of the document. We need three pieces of information: Michelle's certificate (i.e., her public key), the document, and the signature. We now perform two operations. One is to hash the document. The other is to decrypt the signature using Michelle's public key. If the result of these two operations is the same, then we have a valid signature. We now have "proof" that Michelle did in fact sign the document. We want to make an important point here. Anyone with access to Michelle's private key could have signed the document. This is why the private key is preferably known only by Michelle.

Authentication is an application type very similar to digital signatures. Here an entity, perhaps a web server, wants to be able to authenticate that the client sitting at the computer is Michelle. This can be accomplished using public key technology. The web server only needs access to Michelle's public key. The server sends the client, allegedly Michelle, a challenge value. The client encrypts the challenge value using Michelle's private key and sends that result back to the server. We can think of this as Michelle signing the challenge. The server verifies the response by applying Michelle's public key and checking that the result matches the original challenge. The server has then authenticated Michelle.

It is worth emphasizing the sensitivity of the private key in all applications. Suppose Bob is able to steal Michelle's private key. He can decrypt her e-mail, sign documents on her behalf, and authenticate to services as Michelle. In every sense he has stolen her identity within the PKI.

We have described three core applications of public key technology. We have attempted to make clear how the public and private keys are used within these applications. The underlying security in each case is based on possession of the private key.

Server-Centric Private Key Processing

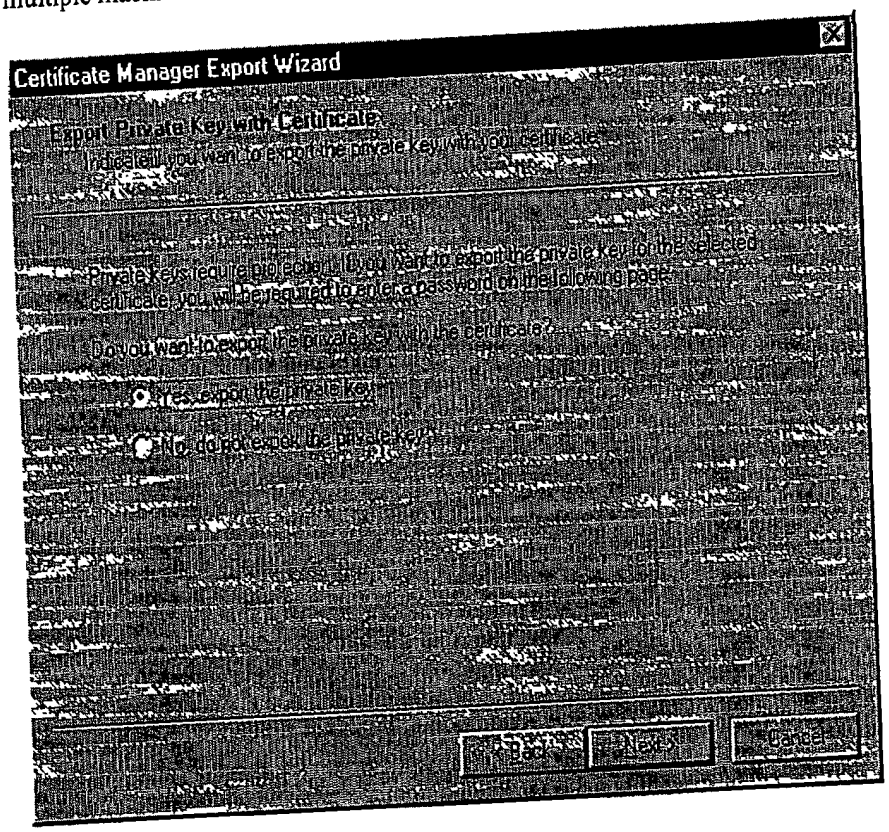
In this paper we discuss the notion of server-centric private keys. We will define what we mean by server-centric private keys. We finish by comparing and contrasting server-centric private keys to the traditional distributed model of private key storage.

Public key cryptosystems are in widespread use today. The most common public key cryptosystems are RSA, Diffie-Hellman (DH), the Digital Signature Algorithm (DSA), ElGamal, and elliptic curve variants of some of these. The systems are also called asymmetric cryptosystems because of the following distinguishing feature. Instead of a single key, these cryptosystems employ a pair of keys. One key, called the private key, is preferably not revealed to anyone except the owner of the key. The other, called the public key, is meant to be revealed to everyone and may even be available in a publicly accessible directory. The public key is frequently packaged together with some identifying information into a digital certificate or just certificate for short. The certificate includes a digital signature from a certificate authority (CA). This signature represents that the CA has made some effort to guarantee that the object or person identified in the certificate does in fact possess the private key corresponding to the public key contained in the certificate. Different CA's offer differing levels of identity checking, including a near-zero level of checking. The more checking, the more expensive the certificate is.

The traditional distributed model of public key cryptography is characterized by the Microsoft implementation in the Microsoft Base Cryptographic Provider 1.0. In this implementation the key pair is generated on the user's computer. The private key is stored in a proprietary format in the registry. The public key is packaged into a PKCS#10 certificate signing request (CSR) and sent to a certificate authority. When the CA signs the certificate it is imported into the registry on the users machine. The private key may be password-protected but this is not enforced. In this model, the greatest measure of security is derived from the local access requirement. The password provides some protection in the event that an adversary somehow gains local access to the machine. Note that local access can be gained by getting physically to the machine or by getting access through some insecurely configured network service or a trojan application.

Access to the private key by applications may be done silently, with user acknowledgement, or with the user entering a password. The user selects these options. Most users will not understand the security significance of choosing one option over another. The result is that different users will protect their private keys with differing levels of security. There is also an option to export your public key certificate. Users will be asked if they want to also export the private key (Figure 1). Many users will not know what to do here. If they choose to export their private key without realizing what they are doing, they may very well compromise the private key accidentally. These computers are likely to be networked. A remote computer can read the registry and if the permissions are not properly set the password-encrypted private key can be read. If the no password or a poor password was chosen the encryption can be "cracked" revealing the private key. It also possible for the user to accidentally delete their certificates and associated private

keys. Finally, these keys are inherently locked to a given machine, but many users use multiple machines.



These are some of the problems that plague the distributed PKI model. One solution that addresses many of these issues is the smartcard. There are several types of smartcards, but the type that solves most of these problems never allows the private key to leave the card. All private key operations are performed on the card itself. A smartcard is inherently portable and does an excellent job of protecting the private key. It is more difficult to accidentally delete the private key. Most smartcards also incorporate a PIN or passphrase to further control access to the card. However, smartcards need a smartcard reader on every machine the user wants to use. Few computers today come with these readers installed so this represents a considerable cost and maintenance burden for an enterprise. Typically, the smartcard manufacturer provides special software, such as their own CSP to replace Microsoft's, to use the smartcard. In some scenarios smartcards are actually fairly easy to steal or borrow temporarily. If the password or PIN is easy to guess, the smartcard can be used to forge transactions in a way that may leave few clues. Finally, smartcards can be lost, stolen, or damaged.

Another solution is the server-centric model. In this model the private keys are stored in a central location accessible over the network. This is the model Who?Vision is implementing. As of now the SigningEngine™ is the only service Who?Vision is offering that uses the server-centric private key storage. This service combines security,

portability, availability and simplicity to generate non-repudiatable, legally binding digital signatures.

The SigningEngine™ service is integrated into the Who?Vision ethentication system. Preferably, for signed transactions, Who?Vision keeps an audit trail including evidence that can be used to defend against a fraudulent or mistaken attempt to repudiate the transaction. Preferably, this does not include the actual document being signed. SSL is used to protect the transport between the two parties and the TrustEngine™.

Another implementation or embodiment of the server-centric model involves sending the private key information securely to the user's computer when demanded by an application. Here, differing applications store and retrieve private keys in a somewhat proprietary manner. Preferably, the applications use the Microsoft Crypto API. This API also defines a service provider interface (SPI). Microsoft provides an SDK for developers to aid development of these third-party Cryptographic Service Providers (CSP's). Third-party CSP's have been written to implement smartcard-based private key solutions. With such a solution, we should be able to capture every application which user MS CAPI for at least private key operations.

Who? Vision Authentication System

Below is a diagram of the Who? Vision authentication system.

Legend:

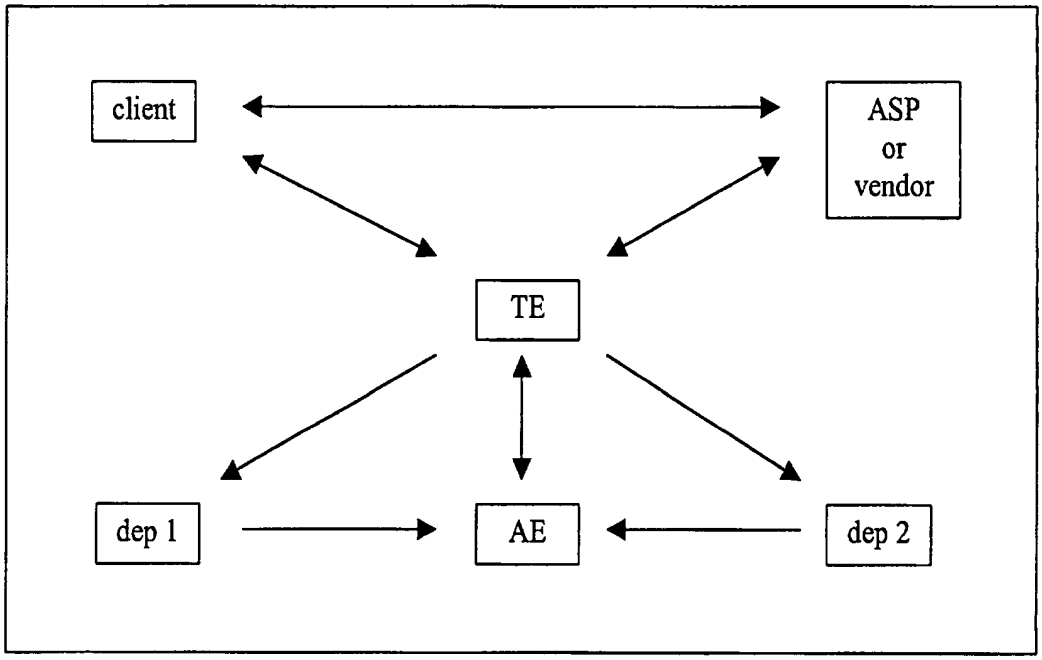
TE is the Trust Engine.

AE is the Authentication Engine.

ASP is the Application Service provider, or more generically the vendor.

dep 1 is depository 1.

dep 2 is depository 2.



These are the steps that occur during an authentication attempt. Enrollment is presented later.

Authentication Data Flow

	Send	Receive	SSL	Action
1	client	ASP	1/2	Transaction occurs, transmits VUID
2	ASP	client	1/2	transmits TID, AV
3	client	ASP	1/2	sends ACK
4				authentication information is collected from user
5	ASP	TE	Full	transmits VUID, TID, ASP, AV
6	TE	ASP	Full	sends ACK
7				TE creates record in audit database, maps VUID to UID
8	client	TE	1/2	VUID, TID, Pub_AE (TID, b')
9	TE	Dx	Full	UID, TID
10				Dx creates record in audit database
11	Dx	AE	Full	TID, Pub_AE(TID, bx)
12	TE	AE	Full	Pub_AE (TID, b')
13	AE	TE	Full	TID, FAV
14	TE	ASP	Full	TID, yes/no
15	ASP	client	1/2	TID, Confirmation message

Legend:
TE refers to the TrustEngine.
AE refers to the Authentication Engine.
VUID is the Vendor Specific User Identification number.
UID is the Who?Vision User Identification number.
TID is the Transaction Identification number.
AV refers to the Authentication Vector.
b' is the authentication information gathered from the user at the time of transaction.
bx is the split component of the secret authentication material held by depository x.
Dx is the x'th depository, where x = 1, 2, or 3.
SSL ½ is server-only authentication, SSL Full is mutually authenticated SSL

The TID is a 192-bit quantity consisting of a 32-bit UNIX timestamp (the number of seconds since midnight, Jan. 1, 1970) concatenated with a 128 bit random quantity called a nonce concatenated with a 32-bit vendor specific constant. The TID serves two purposes. One is to uniquely identify the transaction in the system and the second is to prevent replay attacks.

Explanation of steps

In step 1 the client is attempting to perform an action that requires authentication, for example clicking on the "PURCHASE" button or clicking on an access-controlled URL. This triggers the vendor-side Who?Vision software to ask for the client's vendor-specific identifier or username. The data sent between the Who?Vision software on the client and the Who?Vision software on the vendor site is secured using SSL. Only the vendor side of the SSL connection authenticates itself, using an SSL server certificate. The Who?Vision vendor-side software has access to the private key corresponding to this certificate. We assume that host security measures are sufficient to protect this private key at the vendor site. The client side is unauthenticated, at least via SSL. This is an example of ½-authenticated SSL. The level of authentication is adequate to prevent man-in-the-middle attacks, but the Who?Vision vendor-side software should not trust that the information it is receiving is coming from a valid user; in fact it does not. Instead, in the subsequent steps it forwards information to the TrustEngine and awaits the TrustEngine's response.

In step 2, the vendor-side Who?Vision software generates a transaction identifier (TID) and decides what level of authentication it requires from the client. This authentication request is packaged in something we are calling an authentication vector (AV). The TID and the AV are sent to the client-side Who?Vision software.

In step 3, the client-side Who?Vision software acknowledges receipt of the information sent from the vendor-side software.

In step 4, the client-side Who?Vision software collects the requested authentication information from the user sitting at the client machine.

In step 5, the vendor-side Who?Vision software transmits the VUID, TID, AV, and client IP address to the TE. This is a fully authenticated SSL connection.

In step 6, the TE acknowledges receipt of the information sent from the vendor-side software.

In step 7, the TE maps the VUID to the Who?Vision unique UID. The TE selects the depositories to query. The TE creates a VENDOR_AUTH_REQUEST record in the audit database consisting of the TID, VUID, AV, UID, client IP address, vendor IP address, vendor domain name, TE timestamp, and depositories selected. The TE examines the vendor ID field of the TID. If the vendor ID field is inconsistent with the TE notion of the vendor then the TE rejects the data and creates a REJECT_VENDOR_ID record in the audit database. The TE examines the timestamp field of the TID and rejects the data if it appears stale. "Stale" is defined for the moment to be ± 600 seconds from the TE's own notion of the current time. If the data is rejected a REJECT_STALE record is created in the audit database. If the record appears to be fresh, then it is looked-up in a cache of

recently seen TID's from vendors. If a match is found, then the data is rejected and a REJECT_REPEAT record is created in the audit database.

Prior to step 8, the client is assumed to have collected the requested authentication information that was specified in AV. In step 8, the client sends its VUID, TID, and an authentication material field encrypted in the public key of the AE to the TE. This field consists of the secret authentication material itself together with the TID. The TE creates a CLIENT_ENCRYPTED_AUTH_MATERIAL record in the audit database consisting of TID, VUID, client IP address, vendor IP address, and the encrypted authentication material field. The TE performs a series of checks on TID similar to those performed in step 8 to check for staleness and replay.

In step 9, the TE sends to each of the selected depositories the UID and TID.

In step 10, each of the depositories creates a record in its audit database consisting of the UID, TID, Dx timestamp, TE IP address, the success/failure of the authentication material lookup, and a hash of the retrieved material.

In step 11, the depositories send their split of the secret authentication material, together with TID encrypted in the AE's public key.

In step 12, the TE forwards the encrypted authentication material it received from the client to the AE.

In step 13, the AE returns to the TE a TID together with a filled-in AV. The TE creates an AE_RESULT record in the audit database consisting of the TID, the filled-in AV, and the TE timestamp.

In step 14, the TE returns the TID together with the success/failure of the authentication event corresponding to that TID.

In step 15, the vendor sends a confirmation message along with the TID. This signals the end of the life of the transaction identified by TID.

Enrollment Steps

The picture for enrollment is unchanged, but the actions taken are somewhat different. Here is a brief summary of enrollment. During the enrollment process the user's secret authentication information is bound to useful identifying information and a unique non-secret identifier such as a username. The secret authentication information is stored in a split form in each depository indexed by the username. The identifying information is not stored at the depositories. The splitting occurs at the client, and the splits are encrypted in the public key of the target depository. The user may also be asked to enter some identification information such as name, address, etc. as part of a common set of

identifying information needed for commerce. Additionally, the user may be asked to enter some information that is relevant only to the vendor through which they are enrolling.

Update Steps

From time to time users will need to update their information. Care is taken to insure that the update of information is done securely. A user may change a portion of their secret authentication information if they can first successfully authenticate using the current authentication information. For example, a user can change their password if they can successfully authenticate with the current password. They can change their fingerprint if they can successfully authenticate using the current fingerprint.

If the user's authentication information is lost or stolen, then the user should re-enroll. Procedures and policies may be established which allow recovery of enrollment information upon successful re-enrollment. According to one embodiment, users who can successfully authenticate with their other factors may change the remaining factor.

Use of SSL

According to one embodiment, we preferably use a custom proprietary secure transport protocol. According to another preferred embodiment, we use ½ authenticated and fully authenticated SSL connections to secure the transport layer communications. We use SSL because of its extensive analysis, testing, and acceptance by the security community, its wide availability in products and toolkits, and its name recognition. According to another embodiment, we may modify SSL to make it more efficient. According to yet another embodiment, we may secure individual items at the application level.

As an alternative to SSL, we may also use VPN technology, especially for the links between the TE, depositories, and AE. Our current preference for SSL for these links is due to the nature of these two technologies. VPN implementations are typically deployed at security gateways whereas SSL is more tightly bound to the application.

Preferably, outside of the client, the entities using the authentication system possess an SSL 1024-bit certificate. Advantageously, we provide for a flexible infrastructure in that we do not specify how the certificates are to be generated or who should sign them.

Signing Data Flow

	Send	Receive	SSL	Action
1	client	Vendor	1/2	Transaction occurs, transmits VUID
2	Vendor	client	1/2	transmits TYPE, TID, AV, M
3	client	Vendor	1/2	sends ACK includes what authentication response is available
4				authentication information is collected from user
5	Vendor	TE	Full	transmits TYPE, VUID, TID, AV, h(M)
6	TE	Vendor	Full	sends ACK
7				TE creates record in audit database, maps VUID to UID
8	client	TE	1/2	transmits TYPE, VUID, TID, Pub_{AE}(TID, b', h(M'))
9	TE	Dx	Full	transmits TYPE, UID, TID
10				Dx creates record in audit database
11	Dx	AE/SE	Full	transmits TYPE, TID, Pub_{AE}(TID, bx)
12	TE	AE/SE	Full	transmits TYPE, Pub_{AE} (TID, b',h(M'))
13	AE/SE	TE	Full	transmits TYPE, TID, FAV, h(M), S= $Priv_{AE}$(h(M)) or $Priv_{CLIENT}$(h(M))
14	TE	Vendor	Full	transmits TYPE, RECEIPT=(TID, yes/no, S), S=$Priv_{TE}$ (h(RECEIPT)
15	Vendor	client	1/2	transmits TYPE, TID, Confirmation message

Legend:
TYPE is a message or command type
TE refers to the TrustEngine.
AE/SE refers to the Authentication Engine and/or the SignEngine
VUID is the Vendor Specific User Identification number.
UID is the Who?Vision User Identification number.
TID is the Transaction Identification number.
AV refers to the Authentication Vector.
FAV is the filled-in Authentication Vector, i.e. which authenticators were verified
b' is the authentication information gathered from the user at the time of transaction.
bx is the split component of the client's secret authentication material held by depository x and also possible the client's private key.
Dx is the x'th depository, where x = 1, 2, 3, or 4.
SSL ½ is server-only authentication, SSL Full is mutually authenticated SSL
S=$Priv_{ENTITY}$(h(M)) is the signature of message M using the private key of ENTITY
Pub_{ENTITY}(x) is the encryption of message x using the public key of ENTITY

The TID is a 192-bit quantity consisting of a 32-bit UNIX timestamp (the number of seconds since midnight, Jan. 1, 1970) concatenated with a 128 bit random quantity called a nonce concatenated with a 32-bit vendor specific constant. The TID serves two

SignEngine

The Who? Vision signing engine (SignEngine) will be used to digitally sign transactions in the ethentication system. There are two proposed methods.

In method 1, the SignEngine will contain a Who? Vision private key. After a vendor and a customer agree to the terms of a transaction (U.S. Steel buys 1000 pounds of ore from Mine A), the vendor presents a document containing the terms of sale to the customer. The vendor sends a hash of this document to the SignEngine (via the TrustEngine). The customer reads the document from the vendor, and if he agrees to the terms, he also sends a hash of the document to the SignEngine (via the TrustEngine). The SignEngine compares the hashes. If they match, then the SignEngine encrypts the hash with the Who? Vision private key. The SignEngine returns the result of the comparison and the encrypted hash to the TrustEngine who then returns the information to the vendor. The encrypted hash is stored in the TrustEngine audit database.

This method of the ethentication signing service verifies that the hashes the two parties (vendor and customer) presented to the SignEngine are identical. A party to the transaction cannot modify the transaction after the fact (buy 500 pounds of ore instead of 1000) and claim it is a valid transaction because the original transaction is signed and stored in the ethentication audit system. According to this method, Who? Vision is not signing on behalf of a vendor or customer but is verifying the fact that both parties presented the SignEngine with identical hashes of documents. The hashes are encrypted by Who? Vision for non-repudiation purposes (a party to the transaction can request a copy of the hash from Who? Vision if challenged on the contents of a transaction).

In method 2, the SignEngine will sign documents on behalf of a user by using the user's private key. Who? Vision will generate key pairs for users at enrollment. (We could use Verisign certs.) The private keys will be split and stored in Who? Vision controlled depositories. A hash of the document to be signed is sent to the SignEngine (via the TrustEngine). The TrustEngine queries the depositories for the users private key segments. The depositories send the key segments to the SignEngine where they are assembled into the users private key. The SignEngine encrypts the hash with the users private key. The encrypted hash is returned to the TrustEngine. The TrustEngine returns the encrypted hash to the user where it can be attached to the original document.

This method of the ethentication signing service differs somewhat from method 1. At enrollment we map the level of Who? Vision enrollment to the class of Verisign certificate that we request. A new key pair is generated even if the user already has one since it is difficult to prove that the users existing private key has not been compromised. According to one embodiment, a key pair is preferably generated for every user.

(Another notable issue is that of signing on behalf of a user by using their private key.)

Based on the above, there can be a difference between signing transactions and signing documents.

Depository Process

The following is an exemplary methodology for splitting secrets, such as Biometrics templates, passwords, personal identification numbers, etc., across a number of storage systems.

For example, suppose we have a secret 'S' we want to split. Pick a random value called 'A'. We set 'B' = A XOR S. Note that the equation, S = A XOR B is now available. Place the values A and B in separate depositories to do 2 out of 2. Based on the foregoing, to derive the secret, S, we need only receive A and B.

To do 2 out of 4, split S again with another pair, say 'C' and 'D', such that C is chosen to be a random value and D = C XOR S. We then put the following values in the depositories:

Depository	Values	
1	A	C
2	A	D
3	B	C
4	B	D

One can go through each of the six possibilities of pairs of depositories to verify that any two have enough information to form the secret value, S. Yet, no single depository has enough information to form S. For example, say we have the info from Depositories 1 and 3. Then we obtain S from A XOR B. We can ignore the value C. To do another example, say we have the information from Depositories 1 and 4. Then we obtain S from A XOR B. Note we could also obtain S from C XOR D.

The client module preferably initially obtains secrets and may advantageously randomize which of the sets in the 2 out of 4 paradigm, e.g., AC, AD, BC, or BD, are sent to a give depository.

Features of the TrustEngine Technology

1. Private keys are stored on server and accessed by authenticating user into server.

2. Private keys and authentication data are stored on n multiple servers in such a way that only k of n servers needs to be accessible at any one time, and a rogue employee at any one of the n servers cannot re-create the keys or authentication data.

3. Use of more than one signing engine at a time to execute parallel copies of the signing in order that the outcome of all engines may be compared to detect any faults/fraud on any one of them.

4. Implementation of a cryptographic protocol, such as CSP, where the activating the cryptographic protocol causes a network call back over to the TrustEngine that then fulfills the requested function (e.g., signing) on the server rather than in the client processor or smartcard as would be normal in a cryptographic protocol.

5. Leveraging the fact that if a hash of a document is generated on a machine, only the hash needs to be transferred to any other machine for the document to be signed. Additionally, a similar strategy can be used for file encryption/decryption on a machine - a symmetric key may be generated, the file encrypted, and the symmetric key sent to the other machine for encryption (e.g., with a user's private key) and returned to the original machine for storage.

6. Given that our server holds the client's private keys, we can re-use one single key multiple times to get various certificates, thus saving multiple key generations.

7. In order to control the risk in an authentication transaction, take into account variables such as financial exposure of transaction be authenticated; time of day; IP address; account history; etc. in order to make dynamic decisions about what is the appropriate amount of data that needs to be gained from the user for successful authentication: simple password, complex personal data, fingerprint, etc.

8. The server centric public key storage also allows a user to unlock the keys with a wide number of authentication mechanisms, including passwords, answers to posted questions, and biometrics.

9. In addition, the TrustEngine system supports key recovery, key history, data recovery, certificate revocation lists, on-line certificate status protocol, and the like.

10. The Trust Engine system may also store the keys in split fashion in the different depositories under the same unique user identity. Such server-centric implementations offer a digital signing service (signing documents, notary, etc.)

11. One embodiment of the TrustEngine System is independent of any single biometrics device, e.g., the System may advantageously support a wide variety of standards based biometrics devices.

12. One embodiment of the TrustEngine System supports multi-factor graded authentication that can be customized to support end user's requirements. For example, graded authentication may be set based on dollar value in a transaction based system, or can be used as input for an organizations access control mechanisms.

13. Yet another embodiment of the TrustEngine System is scalable to an infinite number of users.

14. Another embodiment of the TrustEngine System is platform and vendor independent.

15. Yet another embodiment of the TrustEngine System allows customers to choose signing, encrypting, or both, in their implementation of public key infrastructure.

16. Yet another embodiment of the TrustEngine System compares the output of Signing/Authentication engines to detect system faults/compromises. In addition, the TrustEngine System may determine which one of the engines is malfunctioning.

17. Another embodiment of the TrustEngine System allows for the insurance of transactions against repudiation.

18. According to an additional embodiment, the TrustEngine System offers 4th party document archive functionality. For example, after documents are signed, the system may archive the documents for a period of time agreed to in a Service Level Agreement.

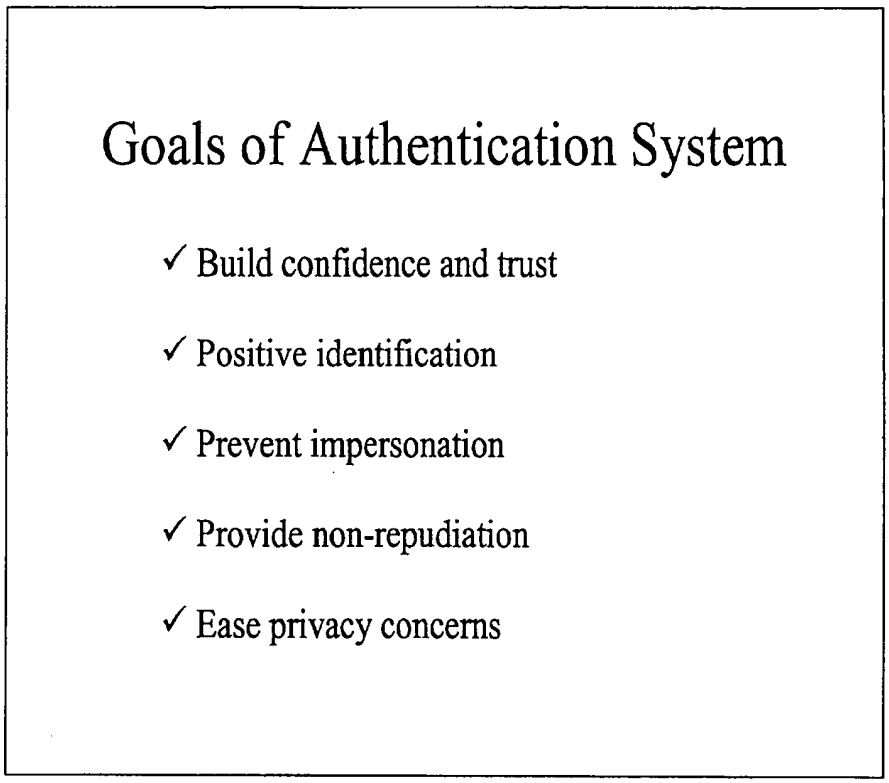
19. According to an additional embodiment, the TrustEngine System supports implementation via mobile platforms such as cell phones or personal digital assistants (PDAs).

20. According to an additional embodiment, the TrustEngine System provides other value-added services such as e-fedex, e-contracts, e-notary.

Security Aspects of the ethenticate™ System

Who? Vision Systems

Good morning. We'd like to describe the security aspects of the ethenticate system. First we'll discuss the goals a successful authentication system. Then we'll talk about the problems that exist in current authentication systems. Next we'll discuss how the ethenticate solution addresses those problems. And lastly we'll go into some details of the ethenticate architecture.



We envision five goals for a successful authentication system:

1. To build confidence and trust in the system for both the user and vendor. Building confidence and trust encourages usage and buy-in of the authentication system.
2. To provide positive identification of individuals. Most commercial transactions require identification of one or both parties.
3. To prevent someone from impersonating anyone in the system. Identification information should be bound to authentication information.
4. To provide non-repudiation capability. Audit trails exist both to prevent repudiation of legitimate transactions and to provide evidence of illegitimate ones.
5. To ease privacy concerns, authentication and identification data is protected against unauthorized disclosure.

Problems with Current Systems

- O Costly to manage and maintain
- O Not all systems are secure
- O Not user-friendly
- O Stronger authentication often needed
- O Privacy concerns

Today vendors manage their own authentication systems.

These can be extremely costly to manage. Each vendor handles its own enrollment and revocation. This cost is duplicated at every vendor.

Furthermore, many of these systems are not secure. Users tend to use the same passwords again and again, and this same sensitive info is on numerous vendor databases. Vendors use varying levels of security to protect this info and the weakest site determines the security of the system.

The user experience is different from vendor to vendor. Furthermore users may have to remember a different password for each vendor.

Most systems are not flexible enough to allow stronger authentication when needed.

Users aren't comfortable with many vendors holding their sensitive identification material that is protected with varying degrees of security.

ethenticate™ Solution

- ✓ Cost-efficient centralized solution
- ✓ Uniform high level of security
- ✓ Simple & consistent authentication
- ✓ Graded authentication levels
- ✓ Secure handling of authentication data

The ethenticate solution addresses each of these problems:

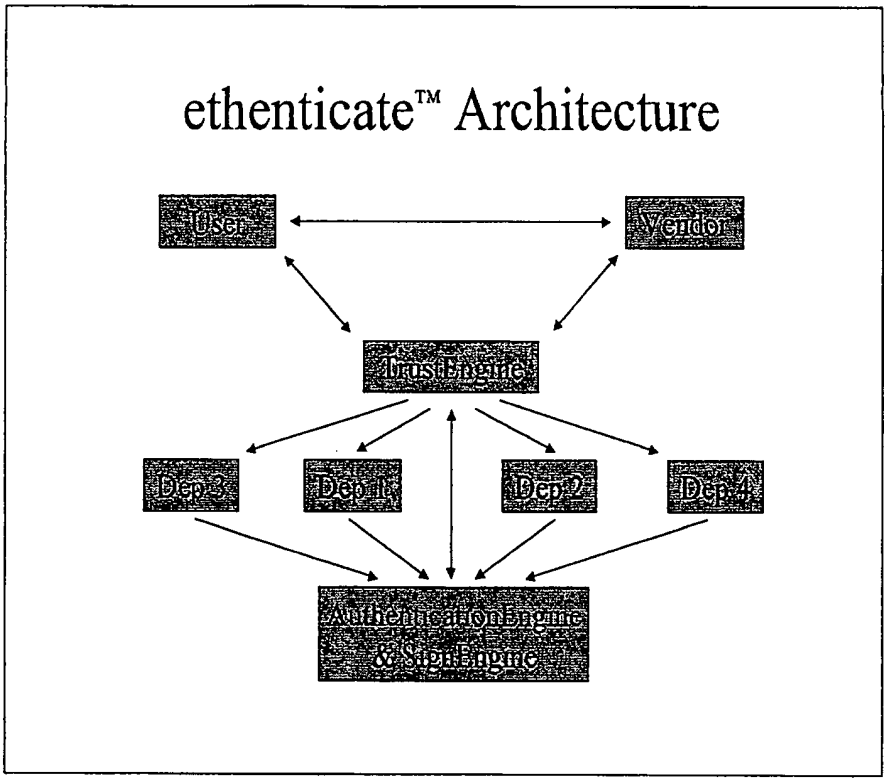
First off, since the ethenticate solution is used by many vendors, the cost of authentication management for the vendor drops significantly.

Secondly, the ethenticate solution is designed to be secure, robust, and reliable. The centralized nature of the ethenticate system makes risk assessment practical and meaningful.

Thirdly, we provide the user and vendor a consistent authentication interface which is simple to use. The user may have a single password in the ethenticate system. Vendors have the flexibility to choose other authentication factors such as a fingerprint scan.

Fourthly, we provide graded authentication levels which can match the authentication level with the value of the transactions. This may be as simple as requiring a password and biometric for high-end transactions.

Finally, according to one embodiment a user's authentication data is transmitted securely, stored securely, and does not leave the ethenticate system. In this embodiment the vendor does not access a user'' authentication material.



(*Only Title is on screen*):

We're going to follow a transaction through the ethenticate architecture.

(*Only User and Vendor boxes are on screen*):

Let's say we have a User about to make an online purchase from a Vendor. An SSL connection is established between the User and Vendor. The vendor requests that the User authenticate via the ethenticate system.

(*Only User, vendor, and TrustEngine boxes are on screen*):

The Users enters his authentication data and that data is sent to the TrustEngine. Simultaneously the Vendor sends message to the TrustEngine to authenticate the User for this transaction. Preferably, communications involving ethenticate components are secured via SSL.

(*Only User, Vendor, TE, and AE/SE boxes are on screen*):

The TrustEngine has the role of traffic cop in the ethenticate system. The TrustEngine forwards the User authentication data to the AuthenticationEngine which will compare this with stored authentication material. Our threat model is preferably that no single entity is completely trusted, including our own components. Hence we have the User wrap his authentication data in the public key of the AuthenticationEngine. So the TrustEngine has no access to this data.